Online Multi-User Workflow Scheduling Algorithm for Fairness and Energy Optimization

Emile Cadorel, Hélène Coullon, Jean-Marc Menaud

Online Multi-User Workflow Scheduling Algorithm for Fairness and Energy Optimization

Emile Cadorel, Hélène Coullon, Jean-Marc Menaud
IMT Atlantique, Inria, LS2N
F-44307 Nantes, France
{emile.cadorel,helene.coullon,jean-marc.menaud}@imt-atlantique.fr

***Abstract*—This article tackles the problem of scheduling multi-user scientific workflows with unpredictable random arrivals and uncertain task execution times in a Cloud environment from the Cloud provider point of view. The solution consists in a deadline sensitive online algorithm, named NEARDEADLINE, that optimizes two metrics: the energy consumption and the fairness between users. Scheduling workflows in a private Cloud environment is a difficult optimization problem as capacity constraints must be fulfilled additionally to dependencies constraints between tasks of the workflows. Furthermore, NEARDEADLINE is built upon a new workflow execution platform. As far as we know no existing work tries to combine both energy consumption and fairness metrics in their optimization problem. The experiments conducted on a real infrastructure (clusters of Grid'5000) demonstrate that the NEARDEADLINE algorithm offers real benefits in reducing energy consumption, and enhancing user fairness.**

***Index Terms*—Cloud Computing, Scientific Workflows, Fairness, Energy, Scheduling, Algorithm**

I. INTRODUCTION

Scientific workflow applications are defined as a set of coarse-grain tasks linked together by data dependencies. Developing scientific applications as a set of tasks with file dependencies is a common pattern, that has been adopted by many users. This approach allows to develop complex applications by dividing it into different simpler parts. Once done, as the parallel parts of the application are highlighted, the application can be executed on a distributed infrastructure.

Historically, most of the academic works done on scientific workflows scheduling have been studied on HPC or Grid infrastructures, where the execution is done directly on physical machines (Bare Metal), with the objective of makespan minimization (*i.e.*, shortest execution time). However, nowadays tasks of scientific workflows are commonly heterogeneous in their libraries and packages dependencies, their data types and even their needed operating systems. For instance, the genomic data stream designed by the ICO in [1] uses data produced by a vendor-specific machine. To convert the output formats of the machine to a standard one, vendor-specific software developed for Windows must be used, while the other applications (*e.g.*, Openswath) must be executed on top of Linux. This heterogeneity is a common issue in scientific workflows. In the last decade, efforts have been conducted for the execution of scientific workflows on Cloud infrastructures, where virtualization mechanisms considerably help in the management of task heterogeneity. However, most of those contributions have been studied in the context of public Clouds that have high availability and limited cost (*i.e.*, pay-as-you-go model), with a cost minimization optimization. Furthermore, most works have focused their attention on the execution of the workflows of one user, and have not taken into account the fairness or equity between users, and as the public Cloud is considered to be managed by a third party, the energy consumption cannot be taken into account. In a public infrastructure multiple users are in competition for a set of resources, on which they submit different applications with uncertain task executions at unpredictable random arrivals. As far as we know there is no work on Grid, HPC or Cloud computing infrastructures that aims at solving the following problematics: (1) uncertainty, that is to say juggle with the unpredictable random arrivals of workflows, and uncertain task execution times; (2) user fairness, in other word, be able to schedule a multi-user workload with a fair sharing of the available resources; (3) energy minimization, *i.e.*, minimizing the energy consumption of the Cloud infrastructure.

In this article, we consider a Cloud infrastructure from the provider point of view with the possibility of provisionning virtual machines (VMs) for two reasons. On one hand, the heterogeneity of operating systems and libraries makes it mandatory to load different operating systems. On the second hand, a multi-user workload is considered, thus, strong isolation, for security reasons, must be guaranteed which is made easier by the VMs. In addition, in this article each workflow is submitted with a deadline. This deadline is used to define the fairness between the users, where an execution is consider fair if all the deadlines of the workflows are guaranteed.

This article presents the following contributions: (1) an online scheduling algorithm for Cloud provider, named NEARDEADLINE, that takes into account both virtualization and deadlines and that schedules the workflows of all users on a set of machines in order to maximize the user fairness and minimize the energy consumption; (2) a detailed evaluation on a real infrastructure execution of the proposed algorithm.

The rest of the document is organised as follows: Section II presentes some noteworthy works related to scientific workflow scheduling. Section III will give a detailed model of the problem. Then Section IV presents our new algorithm NEARDEADLINE, Section V offers a detailed evaluation of our algorithm compared to other algorithms. Finally, the section VI cloncludes this work and opens some perspectives.

II. Related Work

In this section are presented some noteworthy works related to scientific workflows scheduling.

Grid and HPC scheduling algorithm - On the one hand, high-performance computing (HPC) and Grid computing have shown interest in scientific workflow scheduling (heterogeneous coarse-grained, interconnected tasks). These domains do not take virtualization into account in their scheduling algorithms. Indeed, physical machines are directly considered (Bare Metal) and most of the time consolidation of shared resources are not taken into account to avoid interferences. In the following works, we can isolate different groups of algorithms: *batch scheduling*, *list scheduling* and *pack scheduling*.

In [2], Min-Min and Max-Min are presented. Both are batch algorithms (i.e. independent job scheduling). The Min-Min *batch* algorithm first creates a list of independent tasks and schedules them, starting its decision process with the task with the lowest execution time (or the highest execution time for Max-Min). Then it deletes the tasks from the dependency graph, creating a set of new independent tasks. These two phases are repeated until all tasks are scheduled. These algorithms have been designed to plan independent tasks; therefore, they are not well suited for workflows scheduling, as indicated in [3].

In [4], Sun et al. claim that most HPC scheduling algorithms focus on CPU and core usage and that in practice, the user of the batch scheduler must juggle alone with other server resources, such as memory, requiring larger resources (a complete server, for example). Sun et al. consider all resources more precisely by studying two algorithms based on *list* and *pack scheduling*. First, the *list scheduling* algorithm schedules the sorted tasks in a list and tries to minimize the time it takes to complete the submitted tasks (*makespan*). Secondly, the *pack scheduling* algorithm creates packets of tasks that do not exceed the considered capacities and prohibits any further scheduling until all tasks in a packet are completed.

The HEFT algorithm [5] is a well-known heuristic based on a *list scheduling*. This heuristic is divided into two parts. First, a list is created containing all the tasks of the entire workload sorted by priority. Second, each task is scheduled, one by one, on the available resources while trying to minimize the overall completion time required to execute the workflows (*makespan*). In the scheduling phase of HEFT, a selection of physical resources is made. For this purpose, the task completion time is calculated for each physical resource. The one offering the most efficient execution time is selected and the task is scheduled on it.

All the algorithms that have been presented in this subsection aimed at minimizing the makespan of the overall execution of a given workload. They are not well suited for multi-users workloads as they cannot provide any isolation, and cannot consider fairness as they merge all the tasks of all workflows in one big workflow. Additionaly, they are static algorithm that cannot reconsider the current schedule online.

Cloud scheduling algorithm - On the second hand, workflow scheduling strategies have also been studied for Cloud computing infrastructures. Generally those works focus on the case where the Cloud is operated by a third party (*e.g.*, public Cloud providers such as AWS). Thus, the Cloud is considered as an unknown black box that has infinite resources, and that is only limited by the client budget. Typically, as depicted in [6] most of the conducted works focus on minimizing end-user costs to run scientific workflows in a public Cloud where the placement decisions of the different virtual resources are made in an internal scheduler controlled by the Cloud provider. In [3], [7] a budget must be respected according to a public Cloud offer. Such work assumes that the Cloud provider is always able to meet the customer's needs (infinite resources). In [3] an algorithm based on HEFT is presented. This algorithm divides the client's budget by the number of workflows to schedule in a public Cloud environment. The algorithm presented in [3] aimed at minimizing the makespan of the workflows, when respecting the budget of the client. The idea is to remove the aspect of multi dimensional objective optimization, by setting a maximal acceptable value for one of the objective and try to minimize the other.

In [7] a deadline based algorithm to schedule one workflow on IaaS cloud is presented. The IC-PCPD2 algorithm (Iaas Cloud Partial Critical Path with Deadlines Distribution) which was originally dedicated to Grid infrastructure in [8], uses the cheapest resources in priority while trying to respect the deadline of the workflow. In our previous work [9], we have presented the HEFT_*deadline* algorithm that schedules workflows for multiple users in a Cloud infrastructure, while respecting the deadline of each workflows. The objective of this algorithm was to minimize the energy consumption of the infrastructure, by minimizing the number of machines used to execute a given workload.

The algorithms presented in [3], [7], [9] have the same limitation: they do not take uncertainty into account and perform a static schedule that has to be respected. In addition, all the algorithms considering public Cloud environment, such as [3], [7], are not designed for multi-users scheduling problems. Finally, as the physical resources are handled by a third party, the energy consumption cannot be taken into account in the placement and scheduling decisions.

User fairness - The user fairness is defined by a fair sharing of the physical resources between multiple users. As far as we know, there is no work trying to manage the user fairness for workflow scheduling in a Cloud infrastructure, as the Cloud is considered as an unknown black box, and thus the fairness is assumed to be taken into account by the provider. However, some works focusing on the workflow scheduling in a Grid environment, have contributed to enhance fairness between users. In [10], the algorithm shares a set of processors between mutiple workflows and consider a priority on each task. This work assumes that each task consumes exactly one processor and does not consider other resources. Also, in [11], the authors consider the submission of multiple workflows belonging

to multiple users. The resource managment systems of those two works, are difficult to adapt for a Cloud infrastructure.

Uncertainty and online scheduling - Finally some of the algorithms presented above [3], [6] take into account some uncertainty in task execution time. This uncertainty is usually assumed to follow a Gaussian distribution [3], [12], [13]. In addition to this, some recent works perform the scheduling of multiple workflows that are submitted at different unpredictable instants. [12] offers a solution to schedule multiple workflows with unpredictable random arrivals and uncertain task execution times on a IaaS Cloud infrastructure. This work aims at ensuring that the deadline of each workflow is respected, while minimizing the rental cost of the VM in the Cloud infrastructure for a given user. Additionnaly, in [13], a genetic algorithm to solve the scheduling problem of multiple workflows with random arrivals is detailed. These two works, however, do not take into account multiple users, neither fairness. Furthermore, as they consider third party Cloud providers, neither resource consolidation nor energy optimization can be adressed.

To conclude, as far as we know, no existing work aims at solving our specific problem : Defining an algorithm on top of a virtualized infrastructure from the provider side to execute multiple workflows with random arrivals and uncertain task execution times, belonging to multiple users, in order to maximize the user fairness, and minimize the energy consumption.

III. Modeling and Problem Formulation

This section presents a model that describes the scheduling problem: schedule all the tasks of submitted workflows on virtual resources in order to satisfy task dependencies and their resource needs, while respecting the capacity constraints of the physical machines. Our objectives are to maximize the user fairness by respecting the deadline of each users, and to minimize the energy consumption of the physical machines.

Workflow definition - A scientific workflow can be defined as a DAG (Directed Acyclic Graph) $G = (V, A)$ were each vertex $v \in V$ represents a task and each arrow $d \in A$ represents a data dependency between two tasks. For simplicity of further notation, let $\mathcal{J}$ be the set of all tasks composing the submitted workflows over the time. For each task is associated the following properties: (1) its needs in term of computing resources (such as the quantity of memory it requires or the number of cpus, etc.); (2) its software requirements (basically the operating system, libraries and packages needed to execute the task); (3) its execution time represented by two metrics: for a task $j \in J$, μ_j is the number of instructions needed to complete j in average, and σ_j is the standard deviation of the execution time. Such information can be retreived by sampling, and in this paper are assumed to be known.

In addition, a weight representing the size of the data that has to be transferred from a task to another, denoted for each file $a \in A$ between $i \in V$ and $j \in V$ $size_{a,i,j}$, is associated to each arc of the DAGs. It can be noted that a task can transfer the same file to multiple successor tasks, and that a task can get multiple files from the same ancestor task.

Infrastructure definition - The considered infrastructure is composed of multiple nodes (physical machines), denoted $s \in \mathcal{S}$, where $\mathcal{S}$ is the set of all nodes. For each node are attached some properties : (1) its computing capabilities, (the quantity of memory, the number of cpus, etc.); (2) its speed represented by the processor frequency (per CPU), denoted $speed_s$ for $s \in \mathcal{S}$. Let $bw_{s,r}$ be the bandwith between two nodes s and $r \in \mathcal{S}$. One can note that in this paper, as the targeted model only considers a single cluster, the bandwidth is homogeneous for all nodes, thus for sake of simplicity bw will refer to the bandwidth between all nodes, with one exception for the bandwidth between a node and itself.

Any VM can be provisioned on nodes, with any size and with any given operating system. Let $\mathcal{V}$ be the set of all VMs launched on the nodes during the execution from the start. For each vm $v \in \mathcal{V}$, are attached two properties: (1) its hardware capacities, in term of vcpus and memory; (2) its software capacities embedded in its image, basically the installed operating system, packages and libraries. It is ensured that on a given node, at every moment, the number of vcpus and memory used by the hosted VMs does not exceed its capacities. As VMs are used to execute tasks, it is also ensured that at every moment the sum of the needs of the hosted tasks does not exceed the capacities of that VM. In other words, this paper forbids over-provisioning on both nodes and VMs levels. Such capacity constraints can be modeled as a classical bin packing problem. A more formal definition of this problem has been presented in our previous work [9] and is not detailed in this document for sake of space. As over-provisioning is not allowed in this work the deteroration of VM computing speed is assumed to be low [14], and considered to be 5% of the host node speed. Consequently as every task are running in a VM, for simplicity, we will consider that the speed of all node is actually 95% of their real speed.

Scheduling problem - The problem to solve is a dynamic scheduling, meaning that, workflows can be submitted by users at any moment. A workflow $w \in \mathcal{W}$ is submitted at a given time w_{submit}, with a deadline denoted w_{dead}, by a given user w_{user}.

For all tasks $j \in \mathcal{J}$, are defined six temporal notations :

Notation	Meaning
ES_j	The estimated start time of the task
EE_j	The estimated end time of the task
$ETT_{i,j}$	The estimated transfert time between $i, j \in \mathcal{J}$
AS_j	The actual start time of the task
AE_j	The actual end time of the task
$ATT_{i,j}$	The actual transfert time between $i, j \in \mathcal{J}$

Let $j_{succ} \subset \mathcal{J}$ be the set of tasks that succeed $j \in \mathcal{J}$, also called the set of successors. For each task $j \in \mathcal{J}$:

$$\forall s \in j_{succ}, ES_s \geq EE_j + ETT_{j,s} \quad (1)$$

The estimated execution time of a task $j \in \mathcal{J}$ is computed using the execution time properties of the task as described earlier, naming μ_j and σ_j. We consider that the execution time

of a task follow a gaussian distribution $\mathcal{N}(\mu, \sigma)$. Representing the uncertainty of the task execution times by a Gaussian distribution is a common approach in the state of the art [3], [12], [13]. The estimated execution time of a task $j \in \mathcal{J}$ with the Gaussian distribution $\mathcal{N}(\mu, \sigma)$, on a node $s \in \mathcal{S}$ with the uncertainty $x \in]0; 1[$ is as follows:

$$F_x(\mu, \sigma) = \int_{-\infty}^{x} \frac{e^{\frac{-(x-\mu)^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}} \tag{2}$$

$$EE_j = ES_j + \frac{F_x(\mu_j, \sigma_j)}{speed_s} \tag{3}$$

Likewise, after a set of experiments on a real infrastructure we have observed that the VMs boot time is uncertain. Indeed, the boot time of a VM is not static and can differ depending on the stress that is applied to different resources of the node that is hosting it. This observation has been highlighted in many works [15]–[17]. Thus, we assume that the VMs boot time follows a Gaussian distribution composed of two properties, $\mu_{v,s}$ and $\sigma_{v,s}$, respectivally the mean time and the standard deviation of the boot of $v \in \mathcal{V}$ on a node $s \in \mathcal{S}$. These two properties depend on the operating system of a given VM, and the node that host it. Furthermore, they can be retreived by sampling, and are assumed to be known in this paper. One can note that these properties are more static than the task execution time uncertainty, as possible bootable operating systems are bounded and determined by the Cloud provider, while tasks are unknown and chosen by users. Consequently, the administrator can run benchmarks for each OS to get the average boot times of a VM, and its standard deviation.

For each VM $v \in \mathcal{V}$, three different temporal notations are introduced :

Notation	Meaning
EP_v	The estimated instant of the VM provisioning
ES_v	The estimated ready time of the VM
EE_v	The estimated end time of the VM

Let $v_{\mathcal{J}} \subset \mathcal{J}$ for $v \in \mathcal{V}$ be the set of tasks to execute on v, and $s \in \mathcal{S}$ the hosting node for v. For each VM, the following temporal constraints are defined:

$$ES_v = \min_{j \in v_{\mathcal{J}}}(ES_j) \tag{4}$$

$$EE_v = \max_{j \in v_{\mathcal{J}}}(EE_j) \tag{5}$$

$$EP_v = ES_v - F_x(\mu_{v,s}, \sigma_{v,s}) \tag{6}$$

Fairness objective - One of the objective of this work is to ensure the fairness between the different users of the workflow platform. The fairness is represented by the respect of the deadline of each workflow, and thus this objective can be define as the minimization of the sum of deadline violations. The deadline violation of a workflow is defined by Equation (8) and the associated objective by Equation (9).

$$deviation_w = \max_{j \in w_{\mathcal{J}}}(AE_j) - (w_{sub} - w_{dead}) \tag{7}$$

$$violation_w = \begin{cases} deviation_w & If\ deviation_w > 0 \\ 0 & Otherwise \end{cases} \tag{8}$$

$$Minimize(\sum_{w \in \mathcal{W}} violation_w) \tag{9}$$

where $w_{\mathcal{J}} \subset \mathcal{J}$ is the set of tasks of the workflow w, and AE_j is the actual end time of the task $j \in w_{\mathcal{J}}$.

Energy objective - The second objective is the minimization of the energy consumption when answering the execution of all the submitted workflows. It has already been shown [18], [19] that the CPU's energy consumption of a node is not a linear to the CPU load and the duration time. The Figure 1 shows the energy consumption measured on one Ecotype node (see Table II) when executing a variable number of tasks that perform a CPU-burn benchmark.

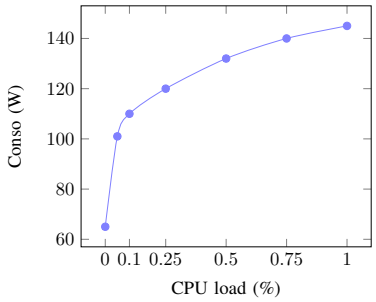

Fig. 1: Power consumption of Ecotype node when varying the number of used cores

It can be noted from Figure 1 that, since CPU consumption is not linear, when using multiple nodes of a cluster, it is more likely that using one node that is 100% used is less consuming than using two nodes that are 50% used, (for example for two Ecotype nodes used at 50% the consumption would be $132 + 132 = 264W$, when with only one node used at 100% and the other at 0%, the consumption would be $145 + 65 = 210W$). For this reason, to optimize the energy consumption, in this work the objective is to use the nodes at the maximum of their capacity by regrouping the tasks on the same nodes.

IV. Deadline based dynamic algorithm

In this section, is introduced our new deadline-aware scheduling algorithm NearDeadline. Our algorithm is built such that workflows belonging to different users - each associated to a deadline - can be submitted at any time to the workflow execution platform. The algorithm is launched when new submissions are performed, with a configurable submission window (*i.e.*, schedule all the workflows that have been submitted during a window of t seconds).

In the following, three definitions will be used: (1) the *workload* is the set of all submitted workflows; (2) an *expected schedule* is generated by the scheduling algorithm and contains all the tasks, files and VMs running or expected to run in the future; (3) a *configuration*, generated from a schedule, contains only the tasks, files and VMs that are currently running or need

to be launched or sent immediately. Thus, a configuration does not contain any information about the future. The algorithm takes into account the current *expected schedule* and generates a new one containing the new workflows. This new *expected schedule* is then used to create a new configuration that will be executed.

Priorities and deadlines - When a new workflow is submitted to the platform, the algorithm first computes its rank based on its deadline and its estimated execution time computed from its critical path (*i.e.*, set of tasks responsible for its overall execution time). The rank of a workflow is computed by Equation (10). The workflow with the lowest rank is handled first.

$$rank_w = w_{dead} - \max_{\forall j \in w_{\mathcal{J}}} (priority_j) \tag{10}$$

The priority of a task presented in Equation (11), is used to compute the execution time of the critical path of the workflow. For each task $j \in \mathcal{J}$ its priority corresponds to the partial execution time of the workflow from this task. It is established according to the average completion time of the task t on all possible nodes, denoted $\overline{ET_j}$, as computed in Equation (12), as well as its average communication time (between all possible nodes) denoted $\overline{ETT_{j,k}}$, for $j, k \in \mathcal{J}$.

$$priority_j = \overline{ET_j} + \max_{z \in j_{succ}} \left(\overline{ETT_{j,z}} + priority_z\right) \tag{11}$$

$$\overline{ET_j} = \frac{\sum_{s \in \mathcal{S}} \frac{F_x(\mu_j, \sigma_j)}{speed_s}}{|\mathcal{S}|} \tag{12}$$

After computing the rank of each submitted workflow (if multiple workflows are submitted at the same time), the NEARDEADLINE algorithm sorts them by increasing rank, and schedule them one by one. As presented in Algorithm 1, the scheduling of a workflow is made in two different phases, the first one tries to schedule each task of the workflow, as near as possible to their respective deadline. This first scheduling is the function SCHEDULEWORKFLOWND. In the second phase, if this schedule fails, the workflow moves in panic mode and will be scheduled in best effort. The rest of this section follows these two phases.

Algorithm 1 Main part of NEARDEADLINE

```
function NearDeadline(workflows, node)
    Q ← workflows
    while |Q|! = 0 do
        panic ← None
        Q ← SortByRank(Q)
        for w ∈ Q do
            if not scheduleWorkflowND(w_J, nodes) then
                Invalidate(j), ∀ j ∈ w_J
                panic ← w
                break
        if panic ≠ None then
            (Q', P') ← RemoveAllUnder(panic, nodes)
            P ← SortByRank(panic + P')
            Q ← Q + Q'
            for all w ∈ P do
                scheduleTaskBestEffort(w_J, nodes)
```

Scheduling near the deadline - The first scheduling that is performed on a workflow is made by the function SCHEDULEWORKFLOWND, presented in Algorithm 2. The idea is to free resources as most as possible before the deadline in case of future submissions with shorter deadlines. The Figure 2 shows an example illustrating this idea, where a first workflow is submitted with a smooth deadline, and afterwards a second workflow is submitted with a more difficult deadline. The y-axis models an imprecise vision of the load on the infrastructure as the percentage of used resources. One can note in this example that the current workload doesn't need to be reconsidered to place the new workflow.

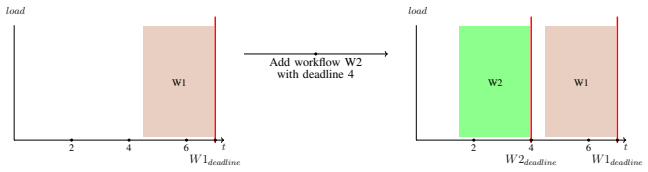

Fig. 2: Scheduling near the deadline to have free resources

Another idea could have been to schedule the workflows using a small amount of resources, as presented in the Figure 3. But as discussed in Section III, the energy consumption is not linear to its executing load. Thus, using this idea would lead to under used nodes during long period of time, wich would be much worse in term of energy consumption than using nodes during small period and leave them in idle mode the rest of the time.

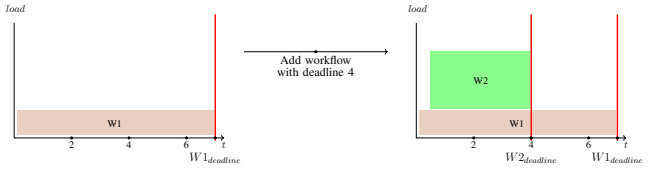

Fig. 3: Scheduling with low resources to have free resources

Algorithm 2 Single workflow scheduling near its deadline

```
function scheduleWorkflowND(tasks, nodes)
    tasks ← SortByBackwardRank(tasks)
    for all j ∈ tasks do
        if not scheduleTaskND(j, nodes) then return False
    return True
```

To perform this scheduling, tasks need to be sorted in backward order (starting by the exit tasks), therefore sorted by backward priority (function SORTBYBACKWARDRANK). The backward priority of a task is computed the same way as the forward priority, but by considering the predecessors instead of the successors. Equation (13) shows the calculation of the backward priority of a task $j \in \mathcal{J}$, where j_{pred} is the list of predecessors tasks of j :

$$bpriority_j = \overline{ET_j} + \max_{z \in j_{pred}} \left(\overline{ETT_{j,z}} + bpriority_z\right) \tag{13}$$

The function SCHEDULETASKND, detailed in Algorithm 3, reserves a location for a task on a node. The function SCHEDULETASKONNODEND, presented in Algorithm 4, is launched

Algorithm 3 Single task scheduling near its deadline

```
function ScheduleTaskND(task, nodes)
    loc ← None
    for all s ∈ nodes do
        loc_s ← ScheduleTaskOnNodeND(j, node)
        if IsBestND(loc_s, loc) then                    ▷ Eq. (9)
            loc ← loc_s
    if loc not None then
        Reserve(loc)
        return True
    else return False
```

on each node for a given task. For a node $s \in \mathcal{S}$ and a task $j \in \mathcal{J}$, this function returns a location, on a VM $v \in s_{\mathcal{V}}$. The returned location is the one that optimizes the local objective evaluated by the fitness function IsBestND, which will be presented in the objective subsection. The goal of this objective is to choose the location that is closest to the deadline.

Algorithm 4 Single task scheduling near its deadline on one node

```
function ScheduleTaskOnNodeND(j, s)
    len ← ComputeLenOfTask(j, s)
    loc ← None
    zero ← currentTimeStamp
    for all v ∈ s_V do
        if v_user = j_user and canRun(v, j) then
            loc_v ← ScheduleTaskOnVMND(j, v, zero, len)   ▷ Eq. (6,4,5)
            if IsBestND(loc_v, loc) then                  ▷ Eq. (9)
                loc ← loc_v
    if loc is None then
        new_v ← EmptyVM(j_os, j_user)
        loc ← ScheduleTaskOnVMND(j, new_v)                ▷ Eq. (6,4,5)
    return loc
```

In the function ScheduleTaskOnNodeND, one can note the called function ScheduleTaskOnVMND. This function retreives the first location where the task can be placed without overcharging the capacity of the VM. This place is searched between the zero instant of the current schedule and the deadline j_{dead} of the task $j \in \mathcal{J}$, of the workflow $w \in \mathcal{W}$ (that is calculated using Equation (14)). This function, detailed in Algorithm 5, is also in charge of the VM dimensions. One may note that a VM can be resized if it is not currently powered on, and therefore the capacity of the node must be taken into account in this function in order to not overcharge it. The resizing of the VM is multi dimensional, it can be either on capacity, and temporal aspects.

$$j_{dead} = \begin{cases} w_{dead} & If\ |j_{succ}| = 0 \\ \min\limits_{\forall z \in j_{succ}} ES_z - ETT_{j,z} & Otherwise \end{cases} \quad (14)$$

As can be seen, the value MES is computed in ScheduleTaskOnVMND. This value is the minimal estimated start time of the task and is computed by Equation (15). Its utility will be explained later, as it requires information given in the panic subsection. CollisionOnVM is the function in charge of checking if the VM, when adding the new task, does not exceed the capacity of the node. There are two different types of VMs collisions. The first one is when a VM cannot be resize in capacity terms, and an example is illustrated in the Figure 4. The second case is when the VM cannot be extended in time because of node overcharge. It is illustrated in Figure 5. CollisionOnVM handles multi-dimensional

Algorithm 5 Single task scheduling near its deadline on one VM

```
function ScheduleTaskOnVMND(j, v, zero, len)
    LST ← j_dead − len                          ▷ Last Start Time, Eq. (14)
    bootTime ← ES_v − EP_v                      ▷ Eq. (6)
    nodeUsage ← NodeUsageWithoutVM(v_host, v)
    nodeCapas ← NodeCapacities(v_host)
    MES ← ComputeMinStart(j, j_pred)            ▷ Eq. (15)
    zero ← max(zero, MES)
    position ← LST
    while position > zero + bootTime do
        loc ← (position, position + len)
        over ← CollisionOnVM(loc, v, nodeUsage, nodeCapas)
        if over is None then
            return loc
        else
            position ← over − len
    return None
```

capacities (vcpus, vram, disk space, ...), but for simplicity of explanation, the two figures represent vcpu resources only.

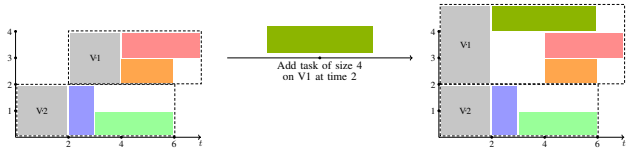

Fig. 4: Collision when increasing the capacities of a VM on a server with 4 cpus

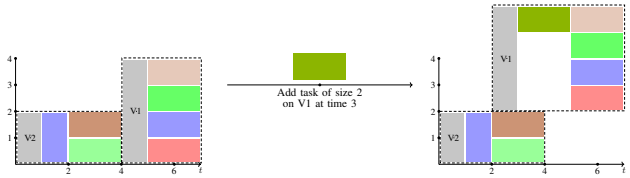

Fig. 5: Collision when increasing the length of a VM on a server with 4 cpus

Let the first case scenario illustrated in the Figure 4, be a *capacity collision*, as the collision occurs inside the VM, and changes the VM capacities; and the second case scenario be a *temporal collision*, as the collision is external to the VM and is due to a duration change on the VM. In the case of a capacity collision, the function CollisionOnVM returns the exact location of the collision inside the VM (in the example, the instant 4). In the case of an temporal collision, the whole VM is considered and the function returns the location of the collision at the node level (in the example, the instant 2). It can be noted that the VM does not always provide a possible location as they are constrained. If none of the currently available VMs hold a valid location, a new empty VM is considered (function EmptyVM, called in Algorithm 4).

Panic mode - When required resources are available on the infrastructure, the previous algorithm ScheduleWorkflowND detailed in the previous subsection performs the scheduling of a workflow near its deadline. However, it may happen that scheduling the workflow in time is impossible with the avaible remaining resources. In that case the considered workflow moves in panic mode, as presented in Algorithm 1.

The function RemoveAllUnder, is an important part of NearDeadline. Since there is no remaining resource to schedule the workflow in time, resources must be released in the current *expected schedule* to be able to find a suitable place

for the new workflow. The function REMOVEALLUNDER explores all the nodes and removes all the non-running tasks belonging to workflows that are less urgent than the new *panic* workflow. The tasks that are currently running, though, are not interrupted for energy and efficiency reasons. We are aware that it can cause missed deadlines when dealing with long-term tasks. However, it would not be difficult to add a system to handle this case by computing the ratio between the remaining execution time and the priority of the tasks, and it would be a possible orientation for a future work. When the function REMOVEALLUNDER ends, it may happen that some running or booting VMs do not contain tasks any longer. In that case, these VMs are killed, even if they have not been useful yet.

Finally, the function REMOVEALLUNDER returns two lists of workflows, a list P' containing all the removed workflows from the *expected schedule* that have already been scheduled in *panic* mode, and the list Q' containing all the other removed partial workflows. It should be noted that those workflows (for both P', and Q') might be uncomplete, as the tasks that are finished and the tasks that are currently running are not rescheduled, or only a subset of the workflow is less urgent than the *panic* workflow. That is to say, when the list Q is refilled, some tasks have ancestor that are still in the *expected schedule*, and therefore it explains why the value MES must be computed in the function SCHEDULETASKONVMND. Each workflow in the list P is then scheduled in best effort mode, using the function SCHEDULETASKBESTEFFORT.

Algorithm 6 Single task scheduling in best effort on one VM

```
function SCHEDULETASKONVMBE(j, v, zero, len)
    MES ← COMPUTEMINSTART(j, j_pred)
    bootTime ← ES_v − EP_v
    nodeUsage ← NODEUSAGEWITHOUTVM(v_host, v)
    nodeCapas ← NODECAPACITIES(v_host)
    position ← MAX(zero, MES)
    while True do
        loc ← (position, position + len)
        over ← COLLISIONONVM(loc, v, nodeUsage, nodeCapas)
        if over is None then
            return loc
        if ISTEMPORALCOLLISION(over) and (position+len) >= EE_v then
            return None
        else
            position ← over − len
```

The function SCHEDULETASKBESTEFFORT is close to the HEFT algorithm of the related work, but with three main differences: (1) it is adapted to take into account virtual resources; (2) it plans workflows one by one instead of mixing all the tasks of all workflows and then executing the scheduling; (3) it uses the fitness function ISBESTBE to choose between multiple possible locations. The function SCHEDULETASKONNODEBE, is used in SCHEDULETASKBESTEFFORT to find a location on a node for a task.

The function SCHEDULETASKONNODEBE, is equivalent to the function SCHEDULETASKONNODEND. There is only two differences. First, instead of ISBESTND the fitness function ISBESTBE is used, this function will be detailled in the objectives subsection. The second difference is the call to the function SCHEDULETASKONVMBE, instead of the function SCHEDULETASKONVMND. This function retreives the first location where the task can be placed without overcharging the capacity of the VM. Unlike the function SCHEDULETASKONVMND, for a task $j \in \mathcal{J}$ and a VM $v \in \mathcal{V}$ this function minimizes the makespan (the minimal EE_j), and hence searches the location starting by the minimal ES_j of the task, defined in Equation (15). Algorithm 6 presents this function.

$$MES_j = \max_{\forall s \in j_{pred}} (EE_s + ETT_{s,j}) \tag{15}$$

The function SCHEDULETASKONVMBE, in Algorithm 6, stops its search when the VM is blocked and cannot grew in length.

Objectives - In the last subsections, two fitness functions ISBESTND and ISBESTBE were introduced. These two functions are used to select one location between two possibilities. The first function, ISBESTND, is used during the first scheduling phase. This function chooses the closest location to the deadline. We assume that, as we are considering the execution time of the task in a pessimist way with the Gaussian distribution, if the scheduling returns a location that is before the deadline, the actual end time of the task will equally be before the deadline. This function gives the priority to the location that is the nearest to the deadline but before it.

The second function ISBESTBE, is the fitness function used when dealing with best effort scheduling. Two different versions of ISBESTBE are available. The first version gives the priority to the VM that owns the greatest number of tasks. The objective is to minimize the number of VMs created on a node, in order to minimize the number of energy waste occuring when a VM is booting. The second version of this fitness function prioritizes location on a VM that optimizes the most the ratio between length and height. The length of a VM $v \in \mathcal{V}$ is $EE_v - EP_v$, and the height is the number of vcpus it requires v_{vcpus}. Equation (16) is used to compute the ratio. In this second version of ISBESTBE, the location in the VM with the highest ratio is selected. If two ratios are equivalent, the earliest location is returned. The objective of this second version is to remove a defect of the first version that can use a single VM with only one vcpu when the load is busy and the VMs very constrained. However, it has the tendency to create more VMs, and therefore consumes more energy.

$$\forall v \in \mathcal{V}, ratio_v = \frac{v_{vcpus}}{EE_v - EP_v} \tag{16}$$

As two versions of the fitness function ISBESTBE are defined, it is possible to define two versions of the algorithm NEARDEADLINE, the first one will be called simply NEARDEADLINE, and the second one using the ratio fitness function will be called NEARDEADLINERATIO.

V. EVALUATION

To compare the performances of the NEARDEADLINE algorithm, we have conducted expriments on a real infrastructure. Those experiments were performed on the Seduce platform [20], a scientific testbed integrated to Grid'5000, that

monitor the electrical consumption of the nodes described in Table II. This evaluation section shows comparisons between NEARDEADLINE, NEARDEADLINERATIO, HEFT [5], and HEFT_DEADLINE [9]. Both HEFT and HEFT_DEADLINE are presented in the Section II. As the algorithms proposing online workflow scheduling with uncertainty in the state of the art consider the Cloud resources as a black box handled by a third party, they cannot be adapted in our context without strong modifications. Consequently our evaluation does not present experimental comparison with them.

Execution platform - For this evaluation, we implemented a platform using the library Scala Akka. The architecture of the plaform is modular and allows to indeferently choose one scheduling algorithm or another. Therefore the algorithm comparaisons can be performed in fair conditions.

The platform is based on a Master Worker architecture with three modules: MASTER, SCHEDULER and WORKER. Each node runs an instance of the WORKER daemon that is responsible for the provisioning of VMs on the node, and that answers simple orders such as starting a task on a given VM, downloading a file from another node, etc. The MASTER is in charge of applying the *configurations* that are transmitted to it by the SCHEDULER. To do so the MASTER sends orders to the WORKERS. The SCHEDULER is the entry point of the platform, meaning that the users submit new workflows to it. The SCHEDULER reconsiders its current schedule and generates both a new expected schedule and a new configuration. The VMs provisionned on each node represent computing resources. Thus, they only access the files of their user available on the node hosting them. File transfers are performed at the node level instead of the the VM level. This aims to enhance the overlapping between computations and communications, and hence the efficiency of the execution. The source code of the platform, the different algorithms, and experimental results can be accessed on a public git repository[1].

Experimental workflows and performance metrics - We evaluate the four algorithms by running the realistic workflow *Montage* presented in [21] and depicted in Figure 6. This workflow is a typical case-study used to evaluate scheduling algorithms [7], [12]. It is a complex workflow that integrates most of the workflow classes characterized by Bharati et al. in [22]. The *Montage* workflow used in this evaluation is composed of 31 tasks, and is successfuly executed on the presented results *i.e.*, returning a valid result a the end of each execution.

Table I lists the different tasks of *Montage*, and presents for each task its execution time (on an Ecotype node), its amount of input data and the amount of data it creates. All the VMs launched during the experiments use the Ubuntu 18.04 operating system, with the hypervizor KVM. The image is replicated on each compute node. The properties $\mu_{v,s}$ and $\sigma_{v,s}$, acquired by sampling are set to respectively 31 seconds, and 20 seconds. The certainty x presented in Section III, has been

[1]https://gitlab.inria.fr/ecadorel/workflowplatform

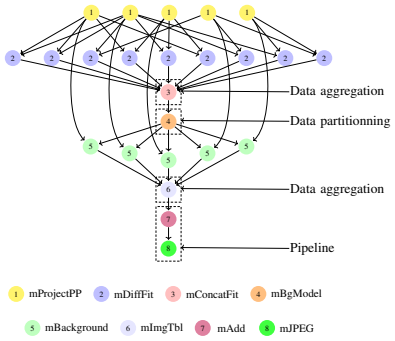

Fig. 6: *Montage* workflow composed of 24 tasks

set to 0.7 for both VM and task execution times. This is the value that gaves us the best results, being a good tradeoff between too optimistic and too pessimistic uncertainty.

We have used 10 nodes of the Ecotype cluster of the experimental platform Grid'5000, presented in the Table II, with one node dedicated to both the MASTER and the SCHEDULER modules, and the others for instances of the WORKER module.

Name	Quantity	μ (s)	σ (s)	size (MB)	input (MB)	output (MB)
mProject	6	23	3	4.5	1.5	8
mDiffFit	3	2	1	18	16	1
mConcatFit	3	2	1	0.5	1	1
mBgModel	3	2	1	0.1	1	1
mBackground	6	2	1	4.4	8	8
mImgtbl	3	2	1	4.5	8	1
mAdd	3	2	1	8.8	16	1
mJPEG	4	2	1	5.1	1	1
Sum	31	37	10	169.5	187	114

TABLE I: Description of *Montage* tasks

The performance metrics are based on the objectives presented in the Section III. The first metric, named *time violation*, is the sum of $violation_w$ of Equation (8) for each $w \in \mathcal{W}$. The second metric, named *nb succeed*, is the number of workflows that have successfuly been executed before their deadline. The last metric, named *power usage*, is the power consumption of the physical machines during a period of 350 seconds, divided by the maximal possible power consumption during the same period of time. The period 350 seconds is the maximal duration of the scenario in our evaluation. The real power consumption of the nodes is retreived thanks to the Seduce platform, which monitors the electrical consumption of the nodes using Power Distribution Units (PDUs).

Scenari - Our evaluations are conducted on a multi-user scenario where each user submits one workflow at a given instant, with a given deadline. We have divided the workload into two subsets, the workload that arrives at time 0, named *initial workload*, and the workload composed of workflows with random arrivals, named the *interfering workload*. A scenario consists in five different variables, where four are constants and the last one is the one that is evaluated in the scenario. The five variables are defined as follows:

- *nb_init*: the number of users submitting a workflow at time 0;
- *nb_inter*: the number of users submitting a workflow at random times;
- *dead_init*: the deadline of the workflows in the initial workload;

Location	Name	Number of nodes	CPU	Memory	Storage	Network
Nantes	ecotype	48	Intel Xeon E5-2630L v4 (Broadwell, 1.80GHz, 2 CPUs/server, 10 cores/CPU)	128 GiB	400 GB SSD	2 x 10 Gbps

TABLE II: Description of the servers of the real infrastructure used in the evaluation

- *dead_inter*: the deadline of the workflows in the interference workload;
- *arrival*: the arrival time of the workflows in the interference workload.

The scenario (A), defined in Table III, aimed at showing the impact of the variation of the deadline of the workflows in the interfering workload, when each interfering workflow arrives each 10 seconds from the instant 0. The second scenario (B) makes the arrival of the interfering workflows vary, by making them arrive all together at a given instant of the scenario. The scenario (C) makes the number of interfering workflows vary when they arrive each 10 seconds. The last scenario (D) aimed at showing the impact when the deadline of the initial workload varies.

Evaluation results - Figure 7 represents the *power usage* metric for the execution of the different scenarios with the different algorithms. The first observation that can be made, is that our algorithm is always able to minimize the energy consumption compared to HEFT. This is due to the dispersion of the tasks among all the nodes. To prevent task dispersion, HEFT_DEADLINE, places the tasks on already used nodes and therefore tries to consolidate before using a new node. One can note that HEFT_DEADLINE is sometimes better than NEARDEADLINE for minimizing energy consumption, as for the scenario (A) for instance. In this scenario, both NEARDEADLINE and NEARDEADLINERATIO reconsider the expected schedule, kill VMs and launched new ones, and therefore consume more energy, when HEFT_DEADLINE will not change anything. However, NEARDEADLINE and NEARDEADLINERATIO are far better than HEFT_DEADLINE in optimizing the user fairness, as it can be seen in the Figure 8 and 9, that respectively shows the *time violation* and *nb succeed* metrics. Figure 8 illustrates that NEARDEADLINE and NEARDEADLINERATIO are able to adapt their expected schedule to the submission of new workflows at differents arrivals. In the scenario (A), both HEFT and HEFT_DEADLINE place the new workflows at the end of their expected schedule, and therefore, the more the deadlines of the interfering workflows are tight, the more the *time violation* will be high. For NEARDEADLINE and NEARDEADLINERATIO, however, this basically does not have any significant impact. The time violation for HEFT and HEFT_DEADLINE are always correlated to the fact that they do not reconsider the previous *expected schedule* in every scenario. The scenario (D) aimed at showing the impact of the variation of *dead_init*. In the fourth variation, for the value 150, no algorithm is able to guarantee the deadline for the initial workload, but there is propably no solution to successfuly perfom this execution in time. In addition, NEARDEADLINE and NEARDEADLINERATIO are still able to overcome HEFT and HEFT_DEADLINE in both energy and user fairness objectives.

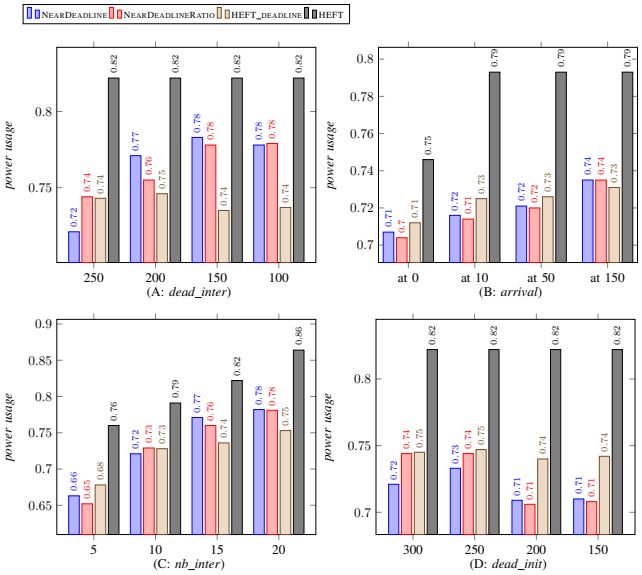

Fig. 7: Power usage of the different scenario executions

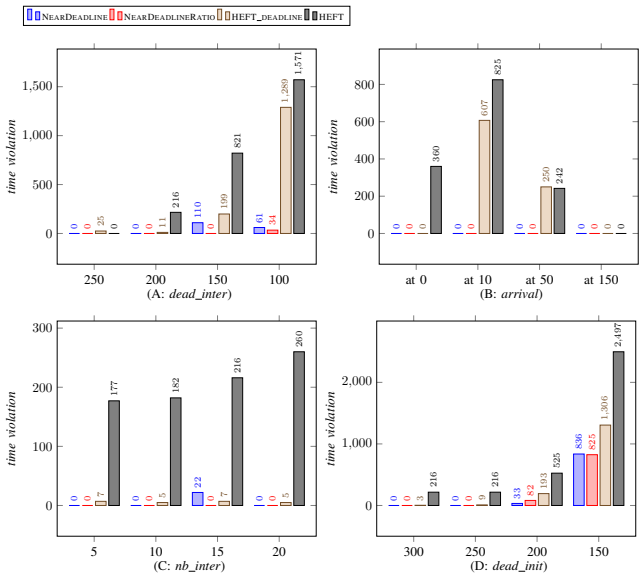

Fig. 8: Time violation of the different scenario executions

The scheduling algorithm is launched in parallel to the workflow execution, and is therefore difficult to evaluate. But one can note that the solving time of NEARDEADLINE and NEARDEADLINERATIO is necessarily small enough for the algorithm to give a *expected schedule* that maximize the user fairness in most case.

VI. CONCLUSION

This paper tackles the problem of scheduling workflows of multiple users with random arrivals and uncertain task execution times, while minimizing the energy consumption of the Cloud infrastructure and maximizing the user fairness.

scenario	*nb_init*	*nb_inter*	*dead_init* (s)	*dead_inter* (s)	*arrival* (s)
A	50	15	300	{250, 200, 150, 100}	each 10
B	50	15	300	200	{at 0, at 10, at 50, at 150 }
C	50	{5, 10, 15, 20 }	300	200	each 10
D	50	15	{ 300, 250, 200, 150 }	200	each 10

TABLE III: Description of the different scenari

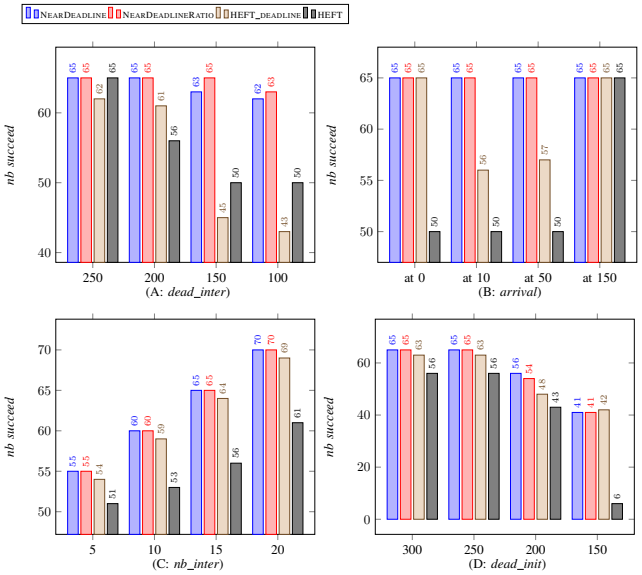

Fig. 9: The number of successful execution under deadline

The NEARDEADLINE algorithm has been presented as a solution to this problem. The NEARDEADLINE algorithm adds deadlines to the workflows chosen by users. These deadlines offer an opportunity to make a smart usage of the infrastructure at disposal while respecting the user initial wishes. The NEARDEADLINE algorithm has been compared to HEFT and HEFT_DEADLINE, on real experiments on a real infrastructure. Experiments have shown real benefits in the reduction of both deadline violation and energy consumption. In future work we plan to take a more complicated infrastructure into account, by considering multiple clusters and heterogeneous network. Furthermore, an economic model should be combined to the deadline mechanism such that energy savings acheived through user flexibility are rewarded.

ACKNOWLEDGMENT

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed (see https://www.grid5000.fr).

REFERENCES

[1] H. L. Röst, Y. Liu, G. D'Agostino, M. Zanella, P. Navarro, G. Rosenberger, B. C. Collins, L. Gillet, G. Testa, L. Malmström, and R. Aebersold, "Tric: an automated alignment strategy for reproducible protein quantification in targeted proteomics," *Nature Methods*, vol. 13, 2016.

[2] J. Yu, R. Buyya, and K. Ramamohanarao, *Workflow Scheduling Algorithms for Grid Computing*. Springer Berlin Heidelberg, 2008.

[3] Y. Caniou, E. Caron, A. K. W. Chang, and Y. Robert, "Budget-aware scheduling algorithms for scientific workflows with stochastic task weights on heterogeneous iaas cloud platforms," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2018.

[4] H. Sun, R. Elghazi, A. Gainaru, G. Aupy, and P. Raghavan, "Scheduling Parallel Tasks under Multiple Resources: List Scheduling vs. Pack Scheduling," Inria Bordeaux Sud-Ouest, Research Report RR-9140, 2018.

[5] H. Topcuouglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, 2002.

[6] E. N. Alkhanak, S. P. Lee, R. Rezaei, and R. M. Parizi, "Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues," *Journal of Systems and Software*, vol. 113, 2016.

[7] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, 2013, including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.

[8] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Cost-driven scheduling of grid workflows using partial critical paths," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, 2012.

[9] E. Cadorel, H. Coullon, and J.-M. Menaud, "A workflow scheduling deadline-based heuristic for energy optimization in Cloud," in *GreenCom 2019 - 15th IEEE International Conference on Green Computing and Communications*. Atlanta, United States: IEEE, 2019.

[10] H. Arabnejad and J. Barbosa, "Fairness resource sharing for dynamic workflow scheduling on heterogeneous systems," in *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, 2012.

[11] R. Ferreira da Silva, T. Glatard, and F. Desprez, "Controlling fairness and task granularity in distributed, online, non-clairvoyant workflow executions," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 14, 2014.

[12] J. Liu, J. Ren, W. Dai, D. Zhang, P. Zhou, Y. Zhang, G. Min, and N. Najjari, "Online multi-workflow scheduling under uncertain task execution time in iaas clouds," *IEEE Transactions on Cloud Computing*, 2019.

[13] F. Deng, M. Lai, and J. Geng, "Multi-workflow scheduling based on genetic algorithm," in *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, 2019.

[14] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. G. Shin, "Performance evaluation of virtualization technologies for server consolidation," 2007.

[15] T. L. Nguyen and A. Lebre, "Conducting Thousands of Experiments to Analyze VMs, Dockers and Nested Dockers Boot Time," INRIA, Research Report RR-9221, 2018.

[16] B. Xavier, T. Ferreto, and L. Jersak, "Time provisioning evaluation of kvm, docker and unikernels in a cloud platform," in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2016.

[17] H. W. et al., "A reference model for virtual machine launching overhead," *IEEE Transactions on Cloud Computing*, vol. 4, no. 3, 2016.

[18] W. Wu, W. Lin, and Z. Peng, "An intelligent power consumption model for virtual machines under cpu-intensive workload in cloud environment," *Soft Computing*, vol. 21, no. 19, 2017.

[19] C. Hsu and S. W. Poole, "Power signature analysis of the specpower_ssj2008 benchmark," in *(IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software*, 2011.

[20] J. Pastor and J. M. Menaud, "Seduce: a testbed for research on thermal and power management in datacenters," in *2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2018.

[21] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Generation Computer Systems*, vol. 29, no. 3, 2013, special Section: Recent Developments in High Performance Computing and Security.

[22] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. H. Su, and K. Vahi, "Characterization of scientific workflows," in *2008 Third Workshop on Workflows in Support of Large-Scale Science*, 2008.